PRÁCTICA 1: DIVIDE Y VENCERÁS

Práctica realizada por:

- Pablo Fernández Valverde

- Martín Reyes Martínez

ÍNDICE

1. Introducción

En esta práctica vamos a realizar un programa con el que calculemos cuales son los 10 mejores jugadores de la historia de la NBA (cuyos datos se obtienen mediante el archivo .csv *NbaStats*, el cual lo cargamos en nuestra clase con un método *cargarArchivo*). Para ello, haremos uso del algoritmo divide y vencerás. Primero se hará uso del algoritmo sin mejora y posteriormente aplicar el mismo algoritmo con una mejora con el fin de reducir el tiempo de ejecución. Finalmente, para demostrar que aplicándole la mejora se reduce considerablemente el tiempo de ejecución, realizaremos una comparativa entre ambos.

Una vez terminado el código, obtenemos los siguientes resultados:

```
LOS 10 MEJORES JUGADORES DE TODOS LOS TIEMPOS SON :

Nombre: Wilt Chamberlain*, Posiciones: [C], Equipos: [PHW, SFW, TOT, PHI, LAL] , con 1153 puntos
Nombre: Kareem Abdul-Jabbar*, Posiciones: [C], Equipos: [MIL, LAL] , con 1076 puntos
Nombre: Michael Jordan*, Posiciones: [SG, SF], Equipos: [CHI, WAS] , con 1075 puntos
Nombre: George Gervin*, Posiciones: [SF, SG], Equipos: [SAS, CHI] , con 1059 puntos
Nombre: LeBron James, Posiciones: [SG, SF, PF], Equipos: [CLE, MIA] , con 1034 puntos
Nombre: Karl Malone*, Posiciones: [PF], Equipos: [UTA, LAL] , con 1005 puntos
Nombre: Karl-Anthony Towns, Posiciones: [C], Equipos: [MIN] , con 965 puntos
Nombre: Kevin Durant, Posiciones: [SG, SF], Equipos: [SEA, OKC, GSW] , con 935 puntos
Nombre: Oscar Robertson*, Posiciones: [PG], Equipos: [CIN, MIL] , con 925 puntos
Nombre: Jerry West*, Posiciones: [PG, SG], Equipos: [LAL] , con 854 puntos

Tiempo total: 4 milisegundos
```

Una vez ya ejecutado el programa, obtenemos los 10 mejores jugadores, ordenados por el total de puntos (calculados mediante la media real), y donde se muestra, primero, todas las posiciones en las que ha jugado durante su carrera, y todos los equipos por los que ha pasado dicho jugador.

2. Explicación de los algoritmos

1. Algoritmo divide y vencerás sin mejora.

```
public static void mejoresJugadoresSinMejora (int inicio, int fin, ArrayList<Player> aux) {
    if (inicio == fin) {
        aux.add(nbaPlayers.get(inicio));
        return;
    }

    int mitad = (fin + inicio) / 2;
    mejoresJugadoresSinMejora(inicio, mitad, aux);
    mejoresJugadoresSinMejora(mitad + 1, fin, aux);
    for (int i = 1; i < aux.size(); i++) {
        Player player = aux.get(i);
        int j;
        for (j = i - 1; j >= 0 && player.getScore() > aux.get(j).getScore(); j--) {
            aux.set(j + 1, aux.get(j));
        }
        aux.set(j + 1, player);
    }
    while (aux.size() > topN) {
        aux.remove(topN);
    }
}
```

2. Algoritmo divide y vencerás con mejora.

```
public static ArrayList<Player> mejoresJugadoresConMejora(int principio, int fin) {
    ArrayList<Player> aux = new ArrayList<Player>(topN);
    if (principio == fin) {
        aux.add(nbaPlayers.get(principio));
    } else {
        int media = (principio + fin) / 2;
    ArrayList<Player> p1 = mejoresJugadoresConMejora(principio, media);
    ArrayList<Player> p2 = mejoresJugadoresConMejora(media + 1, fin);
    int i = 0;
    int j = 0;
    while (aux.size() < topN && i <= p1.size() - 1 && j <= p2.size() - 1) {
        if (p1.get(i).getScore() > p2.get(j).getScore()) {
            aux.add(p1.get(i));
            i++;
        } else {
            aux.add(p2.get(j));
            j++;
        }

    }
    while (aux.size() < topN && i <= p1.size() - 1) {
        aux.add(p1.get(i));
        i++;
    }
    while (aux.size() < topN && j <= p2.size() - 1) {
        aux.add(p2.get(j));
        j++;
    }
 }
    return aux;
}
```

La diferencia primordial entre ambos algoritmos, es que en el algoritmo mejorado hacemos uso de 2 ArrayList<Player> p1 y p2 a los que les aplicamos el algoritmo de forma recursiva a cada uno para que nos realice la ordenación de ambas partes por separado y finalmente a partir de los while vamos realizando la composición, haciendo las comparaciones oportunas para ir metiendo los datos ordenados de ambas partes en un ArrayList<Player> auxiliar. Esto se hará mientras que el tamaño del ArrayList<Player> auxiliar sea menor estricto que el topN.

Estudio del Análisis de complejidad del Algoritmo Divide y Vencerás utilizado en esta práctica:

$$t(n) = \underbrace{a\ t(n/b)}_{\text{Coste Recursivo}} + \underbrace{g(n)}_{\text{Coste NO Recursivo}}$$

$$t(n) = 2t(n/2) + O(n)$$

Donde k = 1

$$n^k \log(n) = n^1 \log(n) = n \log(n) \rightarrow \mathbf{O(n) = n \log(n)}$$

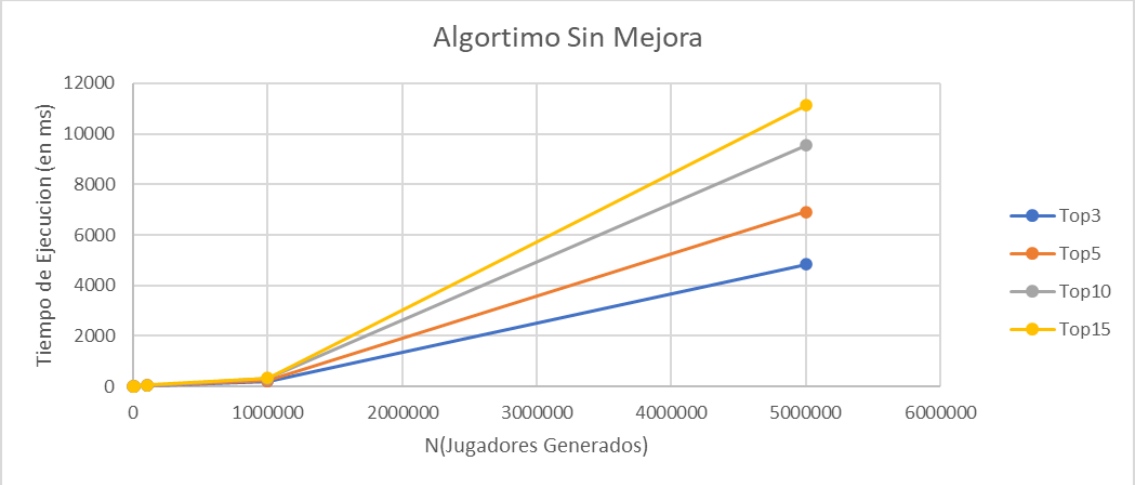
Como conclusión decir que hemos notado que cuando el topN es bajo el orden de complejidad de dicho algoritmo se aproxima a un valor de O(n) = n, pero con forme se va aumentando el valor del topN el algoritmo obtiene un orden de complejidad que se queda establecido con un valor de O(n) = n log(n).

3. Comparativas entre los algoritmos

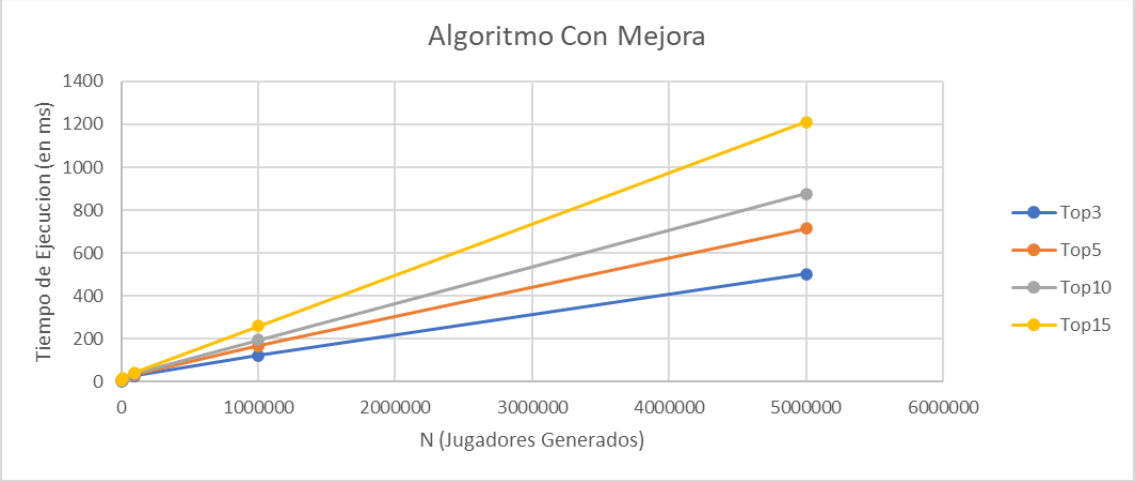
Para llevar a cabo una comparación entre los dos algoritmos, creamos una nueva clase llamada *GenerarJugadoresAleatorios*, cuya finalidad es generar un número de jugadores que nosotros queramos. Esto se debe a que el archivo .csv *NbaStats* no contiene una gran cantidad de datos que nos permita llevar a cabo comparaciones a gran nivel.

```
package Practica01;

import java.util.ArrayList;

public class GenerarJugadoresAleatorios {

    public static void main(String[] args) {
        long inicio;
        long fin;
            createNbaRnd(5000000, 15);
            inicio = System.currentTimeMillis();
            //SolucionPracticaNba.mejoresJugadoresSinMejora();
            SolucionPracticaNba.mejoresJugadoresConMejora();
            fin = System.currentTimeMillis();
            System.out.println();
            System.out.println("Tiempo de ejecución:" + (fin - inicio) + " milisegundos");
    }

    private static void createNbaRnd(int N, int topN) {
        SolucionPracticaNba.nbaPlayers = new ArrayList<Player>();
        for (int i = 0; i < N; i++) {
            Player p = new Player("Jugador" + i, "", "", (int)(Math.random()*N));
            SolucionPracticaNba.nbaPlayers.add(p);
        }
        SolucionPracticaNba.topN = topN;
    }

}
```

Para esto, primero ejecutamos esta clase, sin el algoritmo mejorado, cuando el programa genera 1000, 10000, 100000, 1000000 y 5000000 jugadores aleatorios. Además, para todos estos jugadores aleatorios hemos probado que el programa nos devuelva a los mejores 3, 5, 10 y 15 jugadores, obteniendo los siguientes resultados:



A continuación, hacemos el mismo procedimiento que el anterior para el algoritmo con la mejora:



Analizando los resultados concluimos que gracias a la mejora hemos disminuido notablemente los tiempos de ejecución. Hemos de tener en cuenta que cuantos menos jugadores aleatorios se generan, la diferencia de los tiempos de ejecución entre ambos algoritmos es mínima. Sin embargo, cuando llegamos al punto en que generamos 5 millones de jugadores la diferencia de tiempo está aproximadamente en torno a los 10 mil milisegundos.